Design and Performance of a Wired Viral Network

by

Casey Maloney Rosales Muller

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2003

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 21, 2003

Certified by . . . . . .
Andrew Lippman
Senior Research Scientist, Program in Media Arts and Sciences
Thesis Supervisor

Accepted by . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Design and Performance of a Wired Viral Network

by

Casey Maloney Rosales Muller

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2003, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We constructed a network that fully distributes access to audiovisual information. The information is apportioned among a family of machines. We call it Viral because it can scale in an adhoc way and the addition of new nodes adds to the overall network capability. The network uses multiple multicast for distribution, acquires content from the broadcast television system, and makes viewing video content a more user-centric activity. In this thesis we address the performance of this network in comparison with other ways of providing the same spatio-temporal diversity of access to a body of work. We first provide some theoretical estimate of the capacity, and then we show how the network we built approaches thoses limits. This is done in terms of a presumed distribution of what the user wants.

Thesis Supervisor: Andrew Lippman
Title: Senior Research Scientist, Program in Media Arts and Sciences

Acknowledgments

I would first like to thank my advisor, Andy Lippman, and the MIT Media Lab. The Media Lab is the reason I came to the frigid shores of the Atlantic to study computer science, and Andy has generously given me the opportunity to work for him since my freshman year here. His vision drives our research and his eloquence makes it accessible and interesting.

Next I would like to thank my collaborators on this project, Dimitris Vyzovitis and Ilia Mirkin. Vyzo has been working on parts of this project for his last two Masters degrees and wrote the networking library from scratch, a truly impressive feat. Ilia has been our tireless undergraduate research assistant

I would also like to thank everyone present and past in our research group that I've had the pleasure of working with. Aggelos Bletsas, Dean Christakos, Chris Metcalfe, Maggie Orth, and all the UROPs over the years have made it fun, productive, and challenging to tackle our research topics.

Finally my family and friends, without whom it would be even harder to stay focused.

Contents

List of Figures

Chapter 1

Introduction

The problem we address in this thesis is building a network that provides essentially unlimited access to realtime content and at the same time adapts to new circumstances, and scales with use. Although any data could be deployed and transported on such a network, we selected television as a test example. Television is a good test because its popularity is proven, and examples like TiVo show an evolutionary way to control what you see when. Therefore, we chose as an example a fully distributed TiVo. We call it Viral because the nodes are independent and add capability to the system as they join[18].

Historically, communications has been a centrally developed technology[16]. Both in wires and in the airwaves, it has been regulated and as a result, its development has been restricted. By contrast, the Internet is (one of, if not) the first large-scale network where those practices were violated – it developed in an adhoc manner, under the guidance of an "engineering task force" [39]. Perhaps most importantly, the design placed the intelligence at the nodes rather than in the network itself[23]. There is little state retained in the network, each packet is treated as a separate event; there is no concept of a long-lasting "circuit." A by-product of this design ethos is that the network can freely grow and adapt as uses and technologies change. It is also reliable.

We now consider extending that fundamental design principle to the distribution of realtime, predominantly stored material. As distinguished from the tenets of the Internet itself, we share information storage among all participants in a transaction

and thus localize access to it. Viral networks, as described by Lippman and Reed[18] chart a more personal, neighborly future for communications, and change it from something you buy to something you do. Some aspects of how these ideas could apply to television have been laid down by Vyzovitis[29], but a functioning, useful network has only recently been constructed.

The end goal is to provide realtime access to audiovisual information in such a way that as more users watch "programs", the network capacity appears to grow rather than be consumed. The basic manner by which this is done is by distributing the task of distribution: every node on the network is both a server and a consumer of data.

By-products of this approach are (1) information can be aggregated from different sources to provide increased fidelity[17], (2) the system is reliable in that when a node leaves the network, others can assume the task of bit delivery[10], and (3) the apparent repertoire grows with the number of users. Note that while we experimented with a network that stresses realtime delivery, it can also be used to insure data distribution to a diverse set of recipients, to update distributed archives, and rationalize a distributed database[28].

Similar networks exist and are gaining in prominence, but many are susceptible to legal or technical attack. Napster[49] was popular but legally vulnerable, Gnutella[36] is broken technically, Kazaa[42] is fighting in the courts and struggling with revenue streams, BitTorrent[32] addresses only a small part of the problem, and eDonkey2000[33] has never taken off.

This thesis addresses the mechanics of the network design and the interface we implemented. We embodied the concepts in a system where television is viewed in a new way: it is independent of time and place of original broadcast, and requires no advance intent to "record" a show. Thus, we use the network to place before the user an essentially limitless choice.

Chapter two discusses the definition of a Viral network as propounded by Lippman, and presents examples of the operation of some existing ones[49, 36, 42]. Chapter three presents the design of the underlying structure (Mayhem) and the interface

as constructed. Chapter four presents some analysis that shows the ways by which the network grows with use.

Chapter 2

Viral Networks

Viral networks subvert the traditional notions of infrastructure to create highly scalable communication systems where the intelligence is all in the leaf nodes. Scalability is important, as is the concealment of underlying network details from the end users.

2.1 Motivations for Viral Networks

Communications technology is poised for a fundamental shift rivaling the transition from mainframe to personal computer. Modern telecommunication networks are huge and unwieldy, and have incredible costs associated with new features. Recent research within the Media & Networks Group at the MIT Media Lab[18, 29, 4, 9] has focused on achieving the efficiency needed to build large ad-hoc networks in many domains.

Decentralizing networks and removing initial investment costs places viral networks in the hands of individuals, instead of corporations, and removes the ability of any one group to mandate policy. These technologies, much like the personal computer, are bringing about change at the social and political level as much as the technical and financial ones.

2.2 Description of Viral Networks

The essence of a viral network is that it can be incrementally adopted with benefits at every stage of growth. Lippman and Reed cite the phrase *viral marketing* as inspiration for the naming of this type of system[18]. To achieve this viral adoption pattern, a network must be scalable, decentralized, extensible, bidirectional, and popular. As more nodes join, the overall capability should increase while performance degrades only minimally.

2.2.1 Scalable

Scalability is crucial to the viability of viral networks. If the system does not scale, its own popularity will be its downfall. The most visible example of this type of failure was the first incarnation of Gnutella[36]. The subsequent analysis[22, 1, 26] has provided insight into previously unexamined issues, and makes it clear that the bandwidth required to maintain the network and issue queries can overwhelm in size the file transfer requirements.

2.2.2 Decentralized

Any dependence on a central authority nullifies several important benefits of viral networks. It gives legal attacks a viable target, as in the case of Napster[49], and opens the system up to much simpler denial of service attacks. Decentralization also helps preserve the civil liberties that electronic media are in danger of losing[15], by making such a network be distributed more like printing presses than television.

2.2.3 Extensible

A key point in the philosophy of viral networks is that upgrades to current communication systems are overly complex. Adding new services should be incremental and simple. The system should also be able to evolve around scalability problems that come up or to take advantage of new network topologies and considerations.

2.2.4 Bidirectional

Also key to the design of a viral network is the fact that all nodes can both send and receive content and messages. This is a prerequisite for decentralization, and enables localization, extensibility, and many other useful features.

2.2.5 Popular

By the nature of viral networks, ubiquitous systems are highly functional. This requires that even in the early stages, networks should not constrain the user, and should benefit them early and often. Individuals should be able to see immediate increases in performance as more people join so that they are motivated to sign up their friends.

2.3 Previous Viral Work and Existing Communication Infrastructure

Several experiments with wireless viral networks have been carried out in the Media & Networks group at the MIT Media Lab[4]. Research on wireless viral networks focuses more on the underlying radio technologies, and deals with a different set of issues than a wired one.

Previous communications networks can be seen as viral or non-viral based on their initial cost and rate of growth. In designing our wired viral network we looked at existing systems that have somewhat similar functionality. Although there have been many interesting academically proposed systems such as Chord[27], Tapestry[30], etc, we limit our discussion to large-scale deployed networks where we can discuss the real growth that occurred.

2.3.1 Television

The television network is important because it sets the standard for what users expect a video distribution system to look and feel like. The early days of television were somewhat viral, in that once there was anything being broadcast, users began to sign up, which encouraged more broadcasts and then more users[2]. Cable and satellite television required more infrastructure investment, but were able to leverage the existing sets and programming.

From a technical standpoint, it's almost the exact opposite of the type of network we're trying to create. It's centralized, unidirectional, and traditionally expensive to scale or extend[2].

2.3.2 The Cell Phone Network

The cell phone network is an example of an infrastructure that is non-viral, because until coverage was excellent (an expensive process), many people did not sign up[18]. The technology is centralized, the delays in initiatives such as 3G[31] show that it's difficult to extend, and special efforts must be made to scale with large masses of people[3, 7].

2.3.3 The Internet

The Internet is modeled after many of the principles that we're trying to incorporate into our design. It has evolved to be a supremely extensible medium, and we drew inspiration for many of our techniques from Internet standards. It is also an example of viral growth, because local networks are useful, and become more so when they are interconnected.

On the other hand, important parts of it are organizationally centralized[37], and scaling it up requires a certain finesse.

2.3.4 Napster

Napster[49] is the network that proved that the bandwidth and interest for this type of application is available. However, it's centralized structure proved to be its undoing and the avoidance of similar legal attacks is a central pillar of our design goals.

Its growth was very viral: individuals had incentive to get more users in order to increase the total content and redundancy of the system.

2.3.5 Gnutella

Gnutella[36] in its first incarnation seemed to say that a truly decentralized infrastructure would be difficult to maintain past a certain point of popularity. It emphasized the tradeoff between scalability and centralization.

It also illustrated the problems that can arise from giving too much flexibility to users. Dial-up users used settings far out of line with their bandwidth capabilities, and many people chose to take content without contributing anything back because of the bandwidth problems[1].

Its growth began as viral, but then stopped as it crumbled under its own weight[22]. It has also exhibited the useful viral property of being legally ruled not liable for its content[50].

2.3.6 Kazaa

Kazaa[42], and the FastTrack[34] network overcame several first generation peer-to-peer problems and its communications model is the closest to what we were looking for. It's growth is viral and so far no upper limit on scale has been discovered.

It, like Gnutella-based networks, is also in the process of establishing legal precedent for the immunity of peer to peer networks from prosecution[6].

2.3.7 BitTorrent

A recently deployed file-distribution system is BitTorrent[32]. It is an interesting case because it treats every file as a completely separate network. There is no provision

for queries or discovery within the system, instead this behavior is expected to occur out of band. It basically is targeting the case where a large file is suddenly in very high demand, and is intended to supplement and be enabled by a webserver, rather than replace it.

BitTorrent replaces the standard FTP or HTTP file downloader with its own, which queries a file's "tracker" for hosts that have some or all of the file. It then connects to a set of those, and as it acquires parts of the file, it informs the tracker that it too is now a source. Upon completion of the download, BitTorrent continues to serve file chunks until the user closes the window. As long as one node has a complete copy of the file, downloads can finish, but if no host has the very last section, all other downloaders will hang until a finished user again shares the file. There is an interesting propagating failure in this case, because all downloaders will achieve 99% of the file, but none will finish, making it easy for new nodes to get to 99%, but impossible to complete the download.

2.4 Other Issues

Beyond the design and construction of viral networks, there are several important related concerns. Copyright issues, financial models, and other less technical affairs have a direct effect on engineering choices in the modern political climate.

2.4.1 Copyright Concerns

Any network design to distribute media must be concerned with copyright issues. The RIAA and MPAA have recently begun taking an active hand in prosecuting individuals who violate their members' copyrights. Although copyright is fundamentally a political and social issue, Napster showed that any network that does not consider these ramifications could be shut down. Recent rulings have shown that there are technical solutions that can shield networks from liability, but that leaves the moral question of whether to obey an unjust law or distribute a tool that allows circumvention.

2.4.2 Economic Models

A network like this requires new ways of thinking about revenue and distribution costs. For a solution to copyright concerns to be moral, it must have the permission of the original creators, whether because of compensation or recognition that free distribution serves as good advertising. The network should be flexible enough to allow for multiple solutions along both lines.

2.5 A Wired Viral Network for Video Distribution

As our first large-scale experiment in wired viral networks, we chose to investigate using a viral network to capture and distribute television programs in a more user-centric manner. Television is an application area we have long found interesting. It has significant throughput and latency requirements, forcing non-trivial performance and resources, and it easily scales down to audio or sideways to more general data.

Chapter 3

Design and Construction of a Wired Viral Network for Video Distribution

Our design started from an end-user point of view, and built on the group's previous work with multicast[29, 9, 8]. We examined existing networks that accomplish similar results.

3.1 What The User Should See

Ideally the end-user should have immediate non-linear access to all television ever broadcast, the ability to watch one or more current channels live with pause and rewind, and the ability to do flexible searches through the accumulated content. Simple and advanced user interfaces to the system should coexist peacefully so that the network can be used as a super-powerful but simple personal video recorder, and simultaneously expose powerful functionality for more savvy users.

3.2 An Agent-Based Design

As a starting point in our design, we chose to use an agent-based architecture. So many parts of the system operate with only narrow communication to the rest that we felt this was an efficient choice. It provides clean and simple upgrade paths, the ability to plug in different components into the various parts of the pipeline, and a reasonable way of reusing code.

The program has several straight-forward divisions, each of which can be implemented either as an agent or a collection of agents. One class of agents is responsible for acquiring television content, another for interacting with the user, and then a large class is responsible for accessing, searching, and transporting the actual data.

By establishing the communication model first, we were able to quickly get a framework up and running. As we continued to refine our design, we replaced each agent with more advanced ones. So we started with a crude user interface, a streamer only capable of reading files, and a purely unicast transport system. We were then able to develop easier interfaces, dynamic streamers, and multicast transport agents and drop them in seamlessly. This allowed us to always have a functioning system and make interactive improvements.

3.3 User Interface Guidelines

The simplicity and power of the user interface is important to the smooth acceptance of the network, and since its popularity is its power, this is an important facet to get right.

3.3.1 Time doesn't matter

Even though most of the television content is assumed to come from the cable television network, users should not need to worry about when programs were broadcast if they choose not to. Programs are either recorded, currently playing, or in the future, but the exact times should only matter if the user wants to participate in a shared

television moment live.

Previously recorded television can therefore be displayed to the user in almost any fashion. Saved information about channel and time recorded can be used to construct a traditional program guide that puts the clips within their proper scope. Alternatively, users could become "Video Jockeys" (VJs) and publish television playlists. In essence this allows users to create their own channels, mixing network television with commercial movies, independent media, and anything else.

Future recording could be arranged through guesses based on a purely passive documenting of content accessed, allowing explicit "season passes" as TiVO does[51], or a keyword-based ranking.

Being able to completely remove time from a television interface renders obsolete the concept of "primetime". This empowers the user to override the programming choices of networks and sponsors, and implies that a well marketed show airing during non-peak times could achieve significant viewership from automatically time-shifting consumers.

3.3.2 Tuners don't matter

TV tuner cards for computers have become cheap enough that the channel-saturation point can very easily be reached. At a certain point it will no longer makes sense for broadband users to purchase them, when the number of online nodes with tuners is sufficient to record every channel. Once this threshold ratio of tuners to television channels is achieved in the general community, networking and distributed scheduling can give the appearance of a seamless connection to the cable network. Therefore the user should never need to worry about who actually has tuners.

This is already the case to some extent; with a fast connection it doesn't take much longer to download recently-encoded television from BitTorrent[32] or Kazaa[42] than it would to schedule it to encode with a tuner. This well-organized process only currently occurs for popular series, but it points towards the wholesale process proposed here.

3.4 Constructed User Interfaces

Several interfaces have been written as frontends to the network, some more sophisticated that others. These all use the client library described below to interact with the network, which ensures that changes to the underlying network work seamlessly across different access mechanisms.

3.4.1 Command Shell

The simplest client is merely a command-line interface that's useful for testing purposes, but can also be used effectively for those who prefer minimalist interaction. It can add streams, make queries, and generally ensure that the network is functioning correctly. It can be used to locate keys for standalone players, allowing the lightweight interaction shown in Figure 3-1.

3.4.2 Movie Player

The most trivial useful stream client is simply an open source movie player[47] that has been modified to play "Mayhem" (see below) streams instead of regular files. It has no provisions for finding the key of a stream, but as shown in Figure 3-1, it can be used with the command shell as a simple but effective player. It is also embedded in many of the other interfaces to handle the video decoding.

3.4.3 TV Interface

The first full interface, shown in Figure 3-2 was designed to be somewhat television influenced. Although it uses a mouse and keyboard, and a couple of common icons, it bears little resemblance to a windowed environment, is designed for display on a low-resolution TV, interaction through only a couple of buttons, automatic window optimization, and supports full-screen movie playback. It can make one query at a time and handles preview streams for low-bandwidth sampling of a large number of videos.

Figure 3-1: Modified MPlayer: combined with the shell, no GUI is required

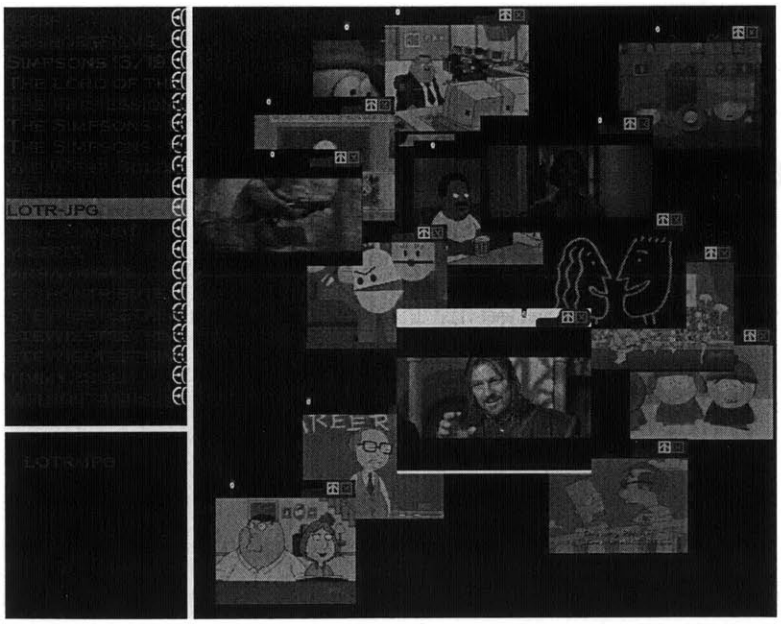

Figure 3-2: StreamBrowser: The TV-friendly interface

3.4.4 Windowed Interface

The next interface looks more like a regular computer application, shown in Figure 3-3. It can handle separate queries in a tabbed interface and also supports preview

streams and a full-screen mode.

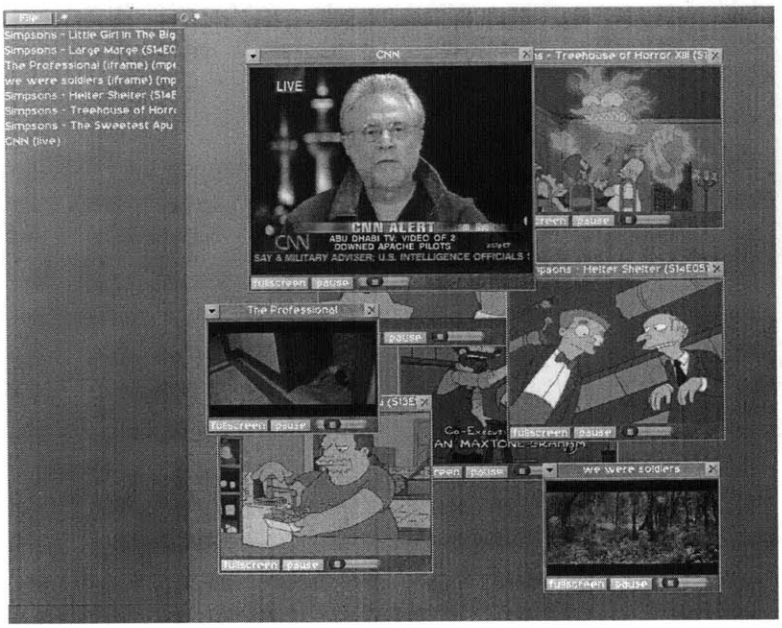

Figure 3-3: VideoGrazer: The windowed interface

3.4.5 Web Interface

The web interface is a minimal webserver that runs locally. Interaction is with a web-browser, with video playing handled by launching the standalone movie player. It's the only interface that presents a full TV guide and allows programs to be recorded.

3.5 Network Design

The underlying network design is where all the really interesting technical research occurs. Our approach requires extreme scalability while still having fairly hefty latency requirements.

3.5.1 Transport

The work of transporting bits builds heavily upon work previously done within the Media & Networks group[29].

Localization is important to successfully scaling our network. Our communication is fundamentally multipoint, so we use multicast[48] whenever possible to attempt to move any given piece of data across any given network link only once.

Multicast has many of the capabilities we need for the low-level distribution of bits. Nodes join well-known control multicast addresses, and then distribution happens on dynamically allocated ones. The Internet Group Management Protocol (IGMP)[40] can be used to exchange group information between routers and ensure that packets are distributed in an efficient manner.

The integral part that multicast plays in IPV6[41] implies that it will have widespread deployment as that standard is migrated to. Until then, it is necessary to use overlay networks either at the network layer (for instance mbone[46]) or in each application[25].

Much work has recently been done on achieving reliable multicast in large networks[19] and using local recovery to aid in late joins[9, 8].

3.5.2 Streams

All data in our network is represented as streams. Each stream has a unique ID that is the SHA-1 hash of the contents of existing files, or placeholders for dynamic ones. A client may fetch a particular key which will attempt to start it flowing across the network, or tap into an existing flow with appropriate catch-up.

Although we wish to keep the data flowing over the network as generic as possible, it is necessary that the networking protocols know about several distinct types of streams. Streams of different types have different IDs, but metadata can indicate other keys that refer to the same content (for instance a live stream that is being recorded into a recorded stream).

Indexed Streams

Indexed streams are programs that have been completely captured and archived. All the data is available somewhere, the total size is known, and we can use the SHA-1

hash of the actual contents as a self-certifying key.

Live Streams

Live streams represent video that is being encoded in real-time. They are small circular buffers that contain only a small amount of time. Although usually these are immediately converted to recorder streams (see below), certain applications like webcams or the Weather Channel may not need any saved history. They are identified by a SHA-1 hash of the channel identifier, and are generally intended only to be accessed locally.

Recorder Streams

Recorder streams are temporary, and represent live streams that are being turned into indexed streams. Unlike live streams, they support seeks within the currently available content, and allow other nodes to watch programs before they are completely recorded. They grow in size until the recording is stopped, at which point they become indexed streams and get a new key based on the SHA-1 hash of their now-static contents.

3.5.3 Queries

Although it is certainly feasible to communicate stream keys in an out-of-band manner, the network also provides facilities for locating local and remote streams by means of a search. Queries are regexps that are sent out in the same way as data packets. They each have a unique ID for result identification (and possibly loop-detection in misconfigured networks) and contain a Time To Live (TTL) field to limit propagation.

The correct way for a node to determine its own stream list is to send out a query that matches any stream with a TTL of 0. This keeps the client interface uncluttered. Queries are commonly sent with increasing TTL values until a suitable result is returned, in an Expanding Ring Search[5].

3.5.4 Messages

For coordination of the agents on the network, we have a generalized message passing framework. Although it is easy to add more, the ones initially received by clients are the following. An example of a session in which an application selects a channel and records it is illustrated in Figure 3-4.

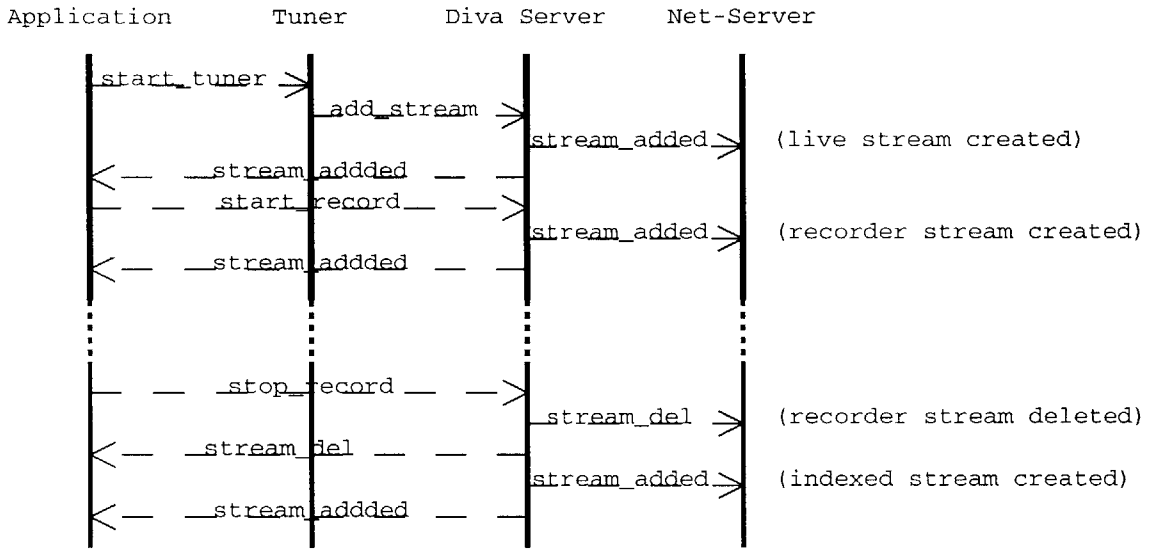

Figure 3-4: Starting a recording: an example of message-based operation

query_result

The result of a query are `query_results` which contain streams that match the parameters in the referenced queryid. Nodes may compare the ID against the list of queries they've sent out to determine if the response is targeted at them, or may simply use the contents to help build a global list, for instance.

stream_available

A successful stream fetch results in a `stream_available` message that indicates data can start to be retrieved from the stream.

fetch_stream_failed

If a stream fetch fails, a `fetch_stream_failed` message is sent to inform the client of the reason.

stream_added

After a new stream is made available for sharing, for example because a file is added, a tuner turned on, or a recorder started, a `stream_added` is sent so that applications can begin using it.

add_stream_failed

If a stream fails to be added, a `add_stream_failed` message is sent to allow the responsible agent to try again.

record_started

This message indicates that a live stream has been successfully converted into a recorder stream, and provides the new key.

record_stopped

This message indicates that a recorder stream has finished, and is now an indexed stream.

start_tuner

This message indicates that a live stream on a specified channel should be started.

stop_tuner

This message indicates that a live stream is no longer needed.

stream_added_notification

This message is sent to agents that ask to be notified about all stream additions, even if they did not request the add.

stream_removed_notification

This message is sent to agents who ask to be notified about all stream removals, even if they did not request the delete.

3.6 Networking Library

The underlying networking library that has been built for this project is called "Mayhem"[45] and tackles the tough problems of scalability, extensibility, and performance that have been laid out in this document. The library takes the form of several servers and a client interface to interact with them. The different layers of the system are illustrated in Figure 3-5.

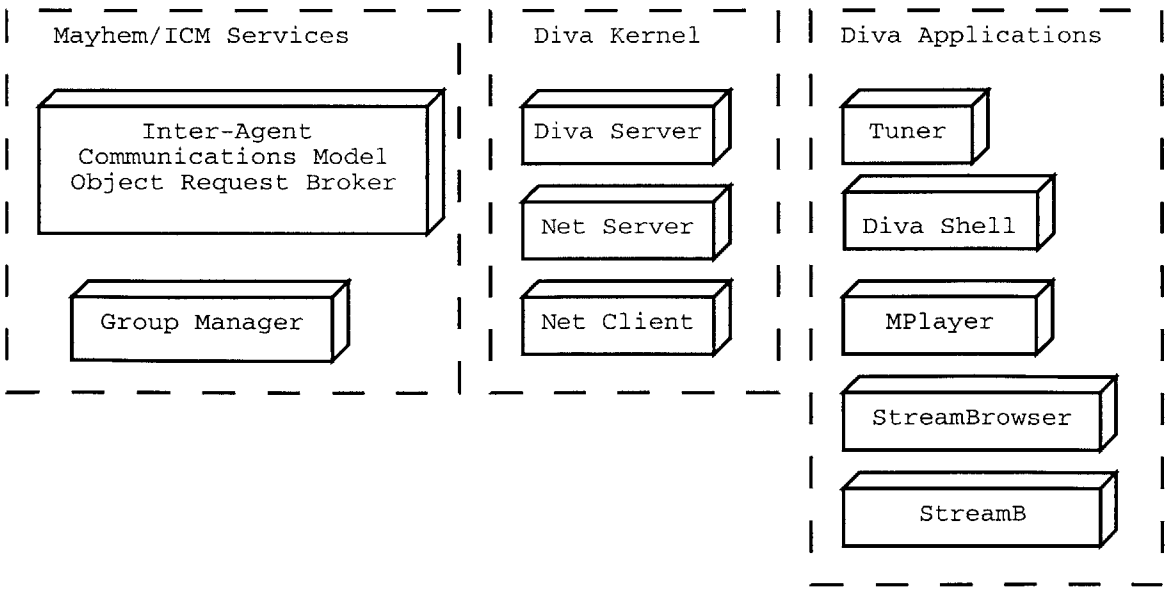

Figure 3-5: Mayhem and Diva Architecture

3.6.1 Client Interface

The client interface provides a simple way to create an application that utilizes the network. When an application initializes the C++ object, it becomes an agent in the viral network and can use the client interface to deal with queries, streams and messages.

3.6.2 Inter-Agent Communication Model (ICM)

The ICM server implements the intra-node messaging architecture and provides the low-level communication path between the agents. It is a reimplementation of a somewhat open protocol[38] with local extensions like access lists and multicast groups.

3.6.3 Group Manager

The group manager handles collections of agents and provides a way to distribute messages to multiple entities.

3.6.4 Diva Server

The Diva server handles requests for the actual video files, and sends chunks of them when asked.

3.6.5 Net Agents

The net agents translate the agent messages to the network and allow inter-node communication.

3.7 Television Recording Agent

The interface between our system and the broadcast TV network is important, if our network is to successfully subvert it. The trickiest question is who controls what channel each tuner records at any time.

On one end of the spectrum is locally-controlled tuners. Under this scheme, each television-connected computer selects what to record based on the local user's preferences and there is no resource optimization. The problem with this scheme is that if everyone with tuners are watching a reality show on FOX, a nearby laptop user is unable to watch Comedy Central even though the other computers could all be sharing only one stream.

On the opposite side is a distributed resource allocation problem, where tuners are used to best serve the needs of the overall network. A decentralized voting scheme could be employed to attempt to record channels that will satisfy as many users as possible. This approach must be carefully thought out, to ensure that it does not deter users with tuners from joining. There must be a way to deal with issues of bad reception, deceptive streams, network segmentation, and changing user interests, and these issues should be dealt with in future work.

3.7.1 Television Interface Guidelines

At the design stage we will leave the issue at simply allowing agents to request a live stream, and allow recording agents to decide the merit of the request. If a web-of-trust infrastructure was developed, each node could then choose to run a simple or advanced television recording agent.

Other issues include recognizing when shows are duplicated on different tuners, and whether it's possible to merge them either whole or in fragments. Proper metadata tagging and recovery from outages are also significant concerns that can luckily be left up to the recording agents.

3.7.2 Constructed Recording Agent

The recording agent handles the encoding of television into a live stream, and also starts the recorder streams when a recording is requested. Currently it only honors local requests, but it could be extended to permit remote tuning when the tuner is otherwise idle.

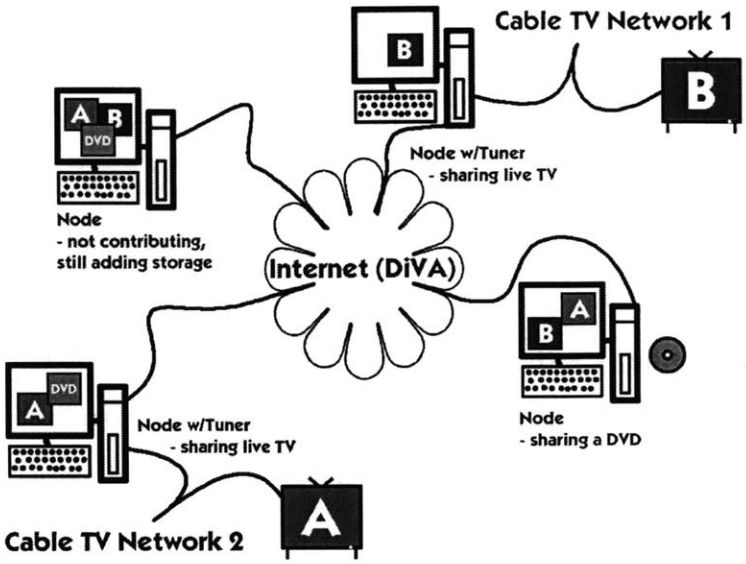

Figure 3-6: A sample network: Nodes with tuners share live streams, all nodes share storage and bandwidth

The tuner utilizes Video4Linux[52] to provide a general interface to tuning cards under Linux[44]. It uses an MPEG library[35] to encode with settings designed for reasonable quality and processor usage given that we're only capturing television.

Chapter 4

Analysis of a Wired Viral Network

The performance of our network is important to its success. Unfortunately it has not yet been deployed to millions of users, so we must rely on models and small installations.

4.1 Expectations

Our goal is that the ability of a user to watch video content is limited only by his local bandwidth. We would like our performance for any given user to be similar to the case where that user is the only consumer and all content is being streamed directly. The biggest hit to this ideal performance is the cost of sharing the user's content out on asymmetric network links. In this respect at least, links that have separate upload and download bandwidths encourage a more equitable sharing to consuming ratio. The lack of universally deployed multicast also complicates achieving this goal, and the widespread use of IPv6[41] will result in a system that more closely approximates viral ideals.

4.2 Testbed Network

Our testbed consisted of three workstations with tuners, five headless servers, and one laptop. The three workstations acquired television content and were used to view

the programs, while the servers were loaded with DVD movies and the laptop served as a frequently-wiped pure consumer.

This testbed was invaluable for early testing, and once the system began to function we were able to get an idea of its performance. Our early performance problems were related to the amount of data that was prefetched for the player. Because of varying bitrates between video content, it was difficult to find a value that provided good performance. The solution is to vary the prefetch value based on a bitrate estimate in the metadata, or failing that the total size if the stream is indexed.

4.3 Extrapolation

The testbed network can be extrapolated to a larger collection of machines and links. We would like to show that performance scales with use, and analyze the benefits that we achieve over alternate distribution schemes. Whenever possible, we've used data from surveys of other systems, particularly Gnutella[24, 21, 20] and assumed a similar distribution of users and content.

4.3.1 Topology

The key difference between our network and others is that we avoid constructing an overlay as much as possible. Arbitrary overlays appear to be the primary cause of bandwidth and latency problems in Gnutella[22], because hosts with low bandwidth and high latency are just as likely to become bottlenecks as better-equipped nodes. The relative success of the FastTrack network is due to the bias towards well-provisioned nodes in selecting SuperNodes[43].

We take this concept a step further by replicating the exact topology of the network links. In locales where multicast is deployed, this is trivial, and we are able to precisely control the flow of data using IGMP[40]. As providers migrate to IPv6[41], this will be the situation across an increasing percentage of the Internet. However, even in multicast-enabled installations, it is reasonable to to assume that some providers will institute access rules and restrictions that will disallow multicast traffic across certain

bandwidth links.

Consider the network shown in Figure 4-1. It represents nearby LANs connected through a central provider. The links directly surrounding n1 can be assumed to be slower than the others. In a network like Napster or Kazaa where peers directly send each other the file, all possible localization advantages are lost, as shown in Figure 4-2. BitTorrent does a better job by using effectively a distributed cache in Figure 4-3, but the localization benefits are haphazard and non-optimal. In Figure 4-4, it can be seen that Mayhem takes advantage of all localization by using only one stream and catching up late joins from the closest available source. Data passes through a link twice only if forced to by nodes going offline.

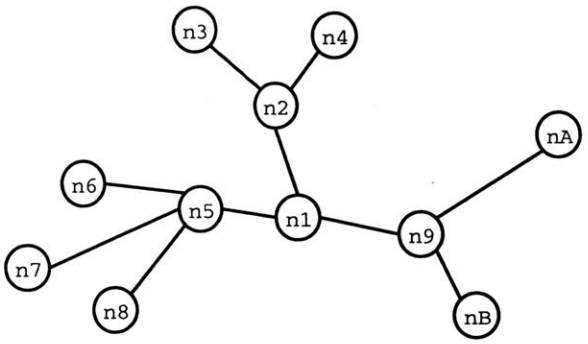

Figure 4-1: Our example network

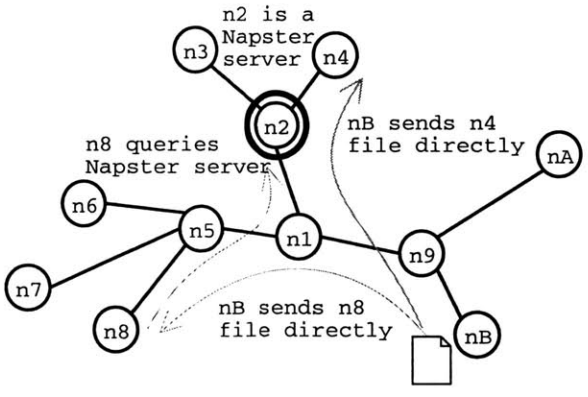

Figure 4-2: A Napster/Kazaa transaction within the network

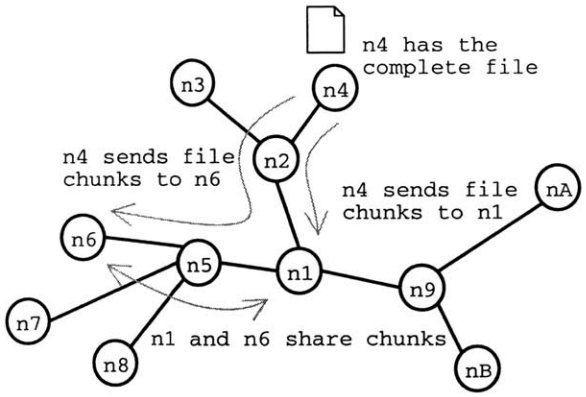

Figure 4-3: A BitTorrent transaction within the network

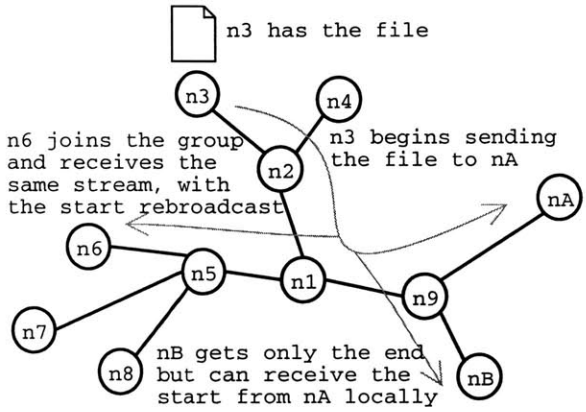

Figure 4-4: A Mayhem transaction within the network

4.3.2 Replication vs. Bandwidth

The fundamental goal of the network is to transport files to nodes that request them. Therefore a crucial question is, how much do we save compared to a straight transfer or overlay peer-to-peer network. Research has shown that the structure of the Internet resembles a power law graph[12]. The extra efficiency of our method over straight transfers is proportional to the number of network links that two paths have in common. If we assume that bandwidths within each domain are much greater than links between them, we can look at just the number of inter-domain paths each stream goes through. From the numbers available[12], it becomes clear than with only a couple of dozen independent users, there must be some path-overlap.

4.3.3 Scalability

Our analysis of Mayhem's scalability must consider three issues: file transfers, network queries, and the cost of maintaining the network topology.

Network Maintenance

The network design does not require any bandwidth to explicitly maintain its structure, but the tradeoff is the overhead implicit in doing multicast routing. Previous work has indicated that these costs are going down, and general multicast routing protocols are scalable[29]. By tying the scalability of this aspect of the system to a general technology, the network can benefit from advanced made in that area. Even in its current state, multicast routing is more efficient in most cases than an expensive overlay network with arbitrary links[22]. As routers become more optimized for multicast, the bandwidth costs should lower even further.

Transfer Efficiency

Multicast transfer of streams allows us to precisely control the flow of bits, with immediate scalability improvements. Consider what would happen in Figure 4-4 if the stream being sent used one third of the bandwidth between nodes $n2$ and $n1$. With Mayhem, nodes $n5$ through nB can still stream the file by joining the multicast group and receiving the initial portion via local recovery. Using any unicast method would saturate the bottleneck link, and even the flexible BitTorrent[32] protocol could end up overwhelming the central links by unnecessary replication of transfers.

Search Efficiency

One of the major contributors to the scalability problems of Gnutella is the query mechanism[22]. It is therefore important to examine the cost of our query mechanism, and how it compares with other techniques. Expanding ring searches, as first presented[5], offer good performance within a localized network. Most of Gnutella's problems stem from the expense of duplicating and propagating queries over widely

varying network links[22], but our queries are multicasted with proper implosion control[29], and do not require duplication. By only slowly widening our search, we keep query messages from going any further than is necessary to find a given piece of content. The downside of this type of query is that longer-range queries will propagate extremely widely, and too many of them will overwhelm the available bandwidth. Once again, the popularity of the network and availability of popular content are crucial to the success of the system.

Searches within Gnutella have been observed to contain many repetitions, enough that a 10 minute cache would have serviced over half of them in all sampled sessions[26]. Because our network is designed around constant television recording and frequent viewing, a 10 minute cache is not unreasonable, and should provide a significant performance boost, particularly on unsuccessful searches. It is reasonable to propose a scheme to always query a certain local radius even with a cache hit, to take maximum advantage of opportunistic caching[29].

4.3.4 Reliability

Another mainstay of distributed systems is the reliability that they confer on their content. Mayhem contains provisions for many local caches, and has actually saved the authors having to reacquire recorded content. Because of the lowering costs of harddrives, it is likely that any content that at least a few nodes have streamed can be found on a longer-term server if uptimes follow the same distribution as Gnutella[24].

4.4 Future Work

There are a multitude of ways that research can proceed, both in technical and non-technical directions.

4.4.1 Accountability and Selfishness

Many of the most interesting extensions to this work concern less technical and more social issues. Although it is mitigated by the use of multicast, the current system allows for a purely selfish user to take advantage of the network's generosity without contributing anything in return[13]. Such a user could disable any use of his upstream bandwidth and fail to contribute even his harddrive space. Because of the way real bandwidth links work, our system does not successfully avoid the "commons" effects as a viral network should. Webs of trust[11, 14] could be used to mitigate this problem, by giving credibility to computers with shared tuners, and having all other users be defined in terms of how trusted they are by the set of tuner-enabled computers. Although this makes it difficult for users to initially use the network and has the potential to complicate the legal issues, a psuedonymous structure could have interesting results and should be investigated.

4.4.2 Advanced Query Support

Expanding ring searches[5] have many positive qualities, but a non-localized caching overlay network could help optimize searches for unpopular content. There are many issues to deal with, but something as simple as a repository of failed ring searches could help deal with the traffic.

4.4.3 Economic Constraints

As mentioned earlier, a network like this forces a rethinking of the economic principles behind copyrighted content. One scheme we've proposed is to have the network collect a fee before authorizing a download of a particular item. This fee would then be split between the copyright holder and the nodes that assisted in the acquisition of the content. By compensating people for their help in downloads, several beneficial effects are achieved. It provides an incentive for users to keep copyrighted material within the system and properly labeled, as they gain nothing by giving the data away for free. It also encourages sharing all possessed content at as high bitrates as possible. If

only the top-level source gets money, behavior that optimizes to share only to sources is invited, so a multi-level marketing payment scheme should be considered. Issues of how to prevent counterfeit content from releasing files they don't own and other authorization issues should also be investigated.

4.5 Conclusion

Although viral technology to some extent is more suited to wireless applications, the wired domain can also benefit by the application of its techniques. Our viral network for video distribution has already changed the way we watch television, and it has the potential to change everything about the medium.

Bibliography

[1] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, September 2000.

[2] Erik Barnouw. *Tube of Plenty: The Evolution of American Television.* Oxford University Press, 2nd revised edition, April 1990.

[3] Elisa Batista. Hello? Salt Lake City, Hello? *Wired News*, February 2002.

[4] A. Bletsas and A. Lippman. Efficient Collaborative (Viral) Communication in OFDM Based WLANs. In *IEEE/ITS International Symposium on Advanced Radio Technologies*, March 2003.

[5] D. Boggs. *Internet Broadcasting.* PhD thesis, Stanford University, 1982.

[6] John Borland. Ruling bolsters file-traders' prospects. March 2002.

[7] Ben Charny. Olympics: A cell phone nirvana? *CNET News.com*, February 2002.

[8] D. Chen. IP multicast video with smart network caches. Master's thesis, Massachusetts Institute of Technology, 1999.

[9] C. Christakos. Taking advantage of distributed multicast video to deliver and manipulate television. Master's thesis, Massachusetts Institute of Technology, 2001.

[10] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM*

Symposium on Operating Systems Principles (SOSP '01), Chateau Lake Louise, Banff, Canada, October 2001.

[11] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proc. of the 9th ACM Conference on Computer and Communications Security*, Washington, DC, USA, November 2002.

[12] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[13] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov, and Mark Lillibridge. Incentives for sharing in peer-to-peer networks. *Lecture Notes in Computer Science*, 2232:75, 2001.

[14] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A reputation system for peer-to-peer networks. In *ACM 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2003.

[15] Ithiel de Sola Pool. *Technologies of Freedom.* Harvard University Press, reprint edition, September 1984.

[16] Ithiel de Sola Pool. *Technologies Without Boundaries: On Telecommunications in a Global Age.* Harvard University Press, October 1990.

[17] R. Kermode. *Smart Caches: Localized Content and Application Negotiated Delivery for Multicast Media Distribution.* PhD thesis, Massachusetts Institute of Technology, 1998.

[18] Andrew Lippman and David P. Reed. Viral communications. Whitepaper, MIT Media Lab, Cambridge, MA, May 2003.

[19] M. Lucas, B. Dempsey, and A. Weaver. MESH-R: Large-scale, reliable multicast transport. In *Proc. IEEE International Conference on Communication (ICC)*, 1997.

[20] Evangelos P. Markatos. Tracing a large-scale peer to peer system: an hour in the life of gnutella. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[21] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. In *IEEE Internet Computing Journal, 6(1)*, 2002.

[22] Jordan Ritter. Why Gnutella Can't Scale. No, Really. http://www.darkridge.com/~jpr5/doc/gnutella.html.

[23] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[24] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, January 2002.

[25] Sherlia Y. Shi and Jonathan S. Turner. Routing in overlay multicast networks. In *IEEE INFOCOM*, 2002.

[26] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. In *The O'Reilly Peer-to-Peer and Web Services Conference*, September 2001.

[27] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM*, August 2001.

[28] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms.* Prentice Hall, 2002.

[29] D. Vyzovitis. An active protocol architecture for collaborative media distribution. Master's thesis, Massachusetts Institute of Technology, Media Arts and Sciences, June 2002.

[30] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[31] 3G. http://www.fcc.gov/3G/.

[32] BitTorrent. http://bitconjurer.org/BitTorrent/.

[33] eDonkey2000. http://www.edonkey2000.com/.

[34] FastTrack. http://www.fasttrack.nu/.

[35] FFmpeg - avcodec. http://ffmpeg.sourceforge.net/.

[36] Gnutella. http://gnutella.wego.com.

[37] The Internet Assigned Numbers Authority. http://www.iana.org/.

[38] Inter-Agent Communication Model. http://www.nar.fujitsulabs.com/icm/.

[39] IETF. http://www.ietf.org.

[40] IGMP (RFC 2236). http://www.ietf.org/rfc/rfc2236.txt.

[41] IPv6. http://www.ipv6.org/.

[42] Kazaa. http://www.kazaa.com.

[43] Kazaa SuperNodes. http://kazaa.com/us/help/faq/supernodes.htm.

[44] Linux. http://www.linux.org.

[45] Mayhem. http://mayhem.media.mit.edu/.

[46] MBONE. http://www.ietf.org/html.charters/mboned-charter.html.

[47] MPlayer. http://www.mplayerhq.hu/.

[48] Multicast (RFC1112). http://www.ietf.org/rfc/rfc1112.txt.

[49] Napster. http://www.napster.com.

[50] Ruling in MGM v. Grokster (courtesy EFF).
http://eff.org/IP/P2P/MGM_v_Grokster/030425_order_on_motions.pdf.

[51] TiVo season passes. http://www.tivo.com/1.2.1.asp.

[52] Video4Linux. http://bytesex.org/v4l/.